

UML 2.0-Based Systems Engineering Using a Model-Driven Development Approach

Dr. Hans-Peter Hoffmann
I-Logix Inc.

More and more, systems engineers are turning to the Unified Modeling Language (UML) to specify and structure their systems. This has many advantages, including verifiability and ease of passing off information to other engineering disciplines, particularly software. This article describes a UML 2.0-based process that systems engineers can use to capture requirements and specify architecture. The process uses the UML exclusively for the representation and specification of system characteristics. Essential UML artifacts include use-case diagrams, sequence diagrams, activity diagrams, state-chart diagrams, and structure diagrams. The process is function-driven and is based heavily on the identification and elaboration of operational contracts, a message-based interface communication concept. The outlined process has been applied successfully at various customer sites. It is assumed that the reader is familiar with the basics of UML.

For many years, software engineers have successfully applied the Unified Modeling Language (UML) to model-based software engineering. There have been several attempts to introduce UML and the underlying object-oriented methods to systems engineering to unify the overall development process. However, many systems engineers continue to use classically structured analysis techniques and artifacts to define system requirements and designs.

One reason for this could be that systems engineering is mostly driven by functional requirements. Speaking in terms of system functionality is still considered the most natural way of expressing a design by most of the domain experts involved in the early phases of system development (electrical engineers, mechanical engineers, test engineers, marketing personnel, and, of course, customers). Given this, the only way to unify the overall development process is to extend UML with regard to function-driven systems engineering, and to define a process that enables a seamless transfer of respective artifacts to UML-based software engineering. The release of UML 2.0 [1] eventually provided the missing artifacts for function-driven systems engineering.

The following sections describe a UML 2.0-based process that systems engineers can use to capture requirements and specify architecture. The approach uses model execution as a means for requirements verification and validation.

Process Overview

Figure 1 shows the integrated systems and software engineering process by means of the classic V. The left leg of the V describes the top-down design flow, while the right-hand side shows the bottom-up integration phases from unit

test to the final system acceptance. Using the notation of state-charts, the iterative characteristic of the process is visualized by the *high-level interrupt* due to system changes. Whenever changes occur, the process will restart at the requirements analysis phase.

Systems engineering is characterized by a sequential top-down workflow from requirements analysis to system analysis and system architectural design. A loop back to the requirements analysis may occur if, during systems analysis or architectural design, functional issues require a reexamination of the higher-level requirements. The software engineering workflow is characterized by the iterative and incremental cycles through the software analysis and design phase, the implementation phase, and the different levels of integration and testing.

It is important to note the creation and reuse of requirements-related test scenarios along the top-down design path. These

scenarios are also used to assist the bottom-up integration and test phases and, in the case of system changes, regression test cycles.

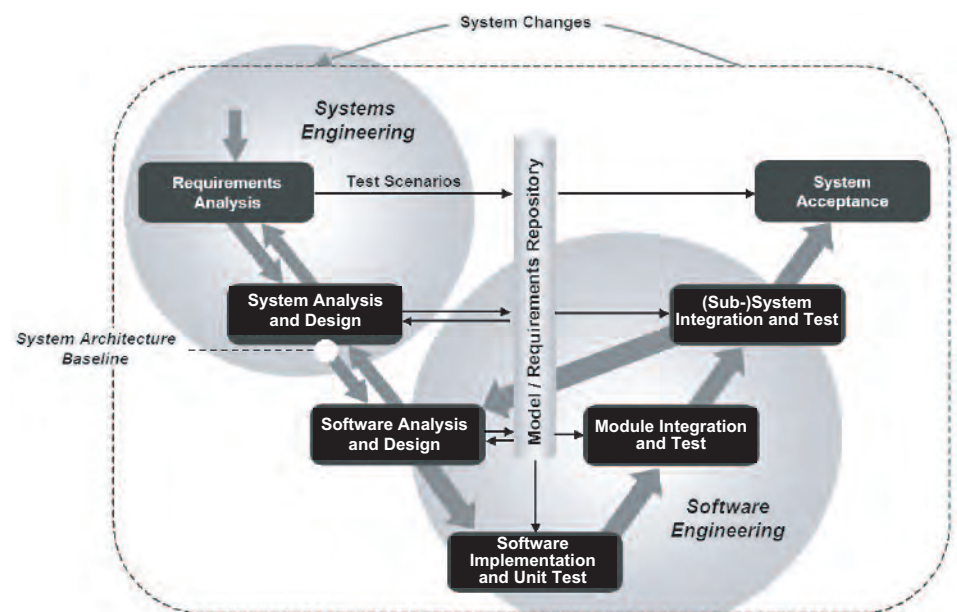
Key objectives of the systems engineering process are as follows:

- Identification and derivation of required system functionality.
- Identification of associated system states and modes.
- Allocation of system functionality and modes to a physical architecture.

Regarding modeling, these key objectives imply a top-down approach on a high level of abstraction. The main emphasis is on the identification and allocation of a needed functionality (e.g., a target tracker), rather than on the details of its behavior (e.g., the tracking algorithm).

Figure 2 depicts an overview of the UML-based systems engineering process. For each of the systems engineering phases, it shows the essential tasks, associated input, and work products.

Figure 1: The Integrated Systems and Software Engineering Process



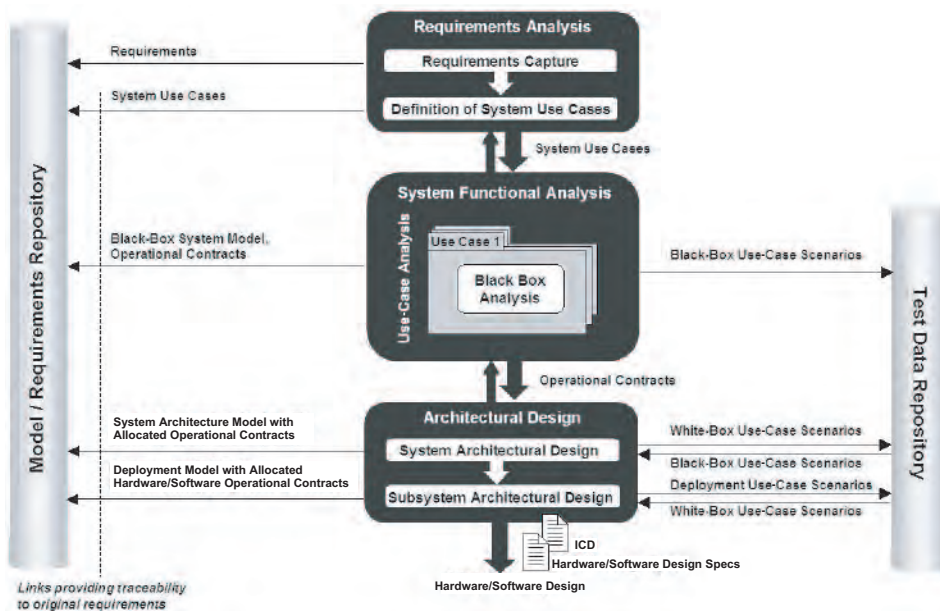


Figure 2: UML-Based Systems Engineering Process

The process is operational contract-driven. An operational contract specifies system behavior by adding pre- and post-conditions to the description of respective operations. Operational contracts are the primary source of traceability to the initial requirements.

The essential tasks in the requirements analysis phase are requirements capture and the grouping of requirements into use cases.

The main emphasis of the system functional analysis phase is on the transformation of the identified functional requirements into a coherent description of system functions (operational contracts). Each use case is translated into a respective black-box model and verified and validated through model execution. Incrementally, these black-box use-case models are merged to an overall black-box system model.

The focus of the subsequent system architectural design phase is the allocation of the verified and validated operational contracts to a physical architecture. The allocation is an iterative process. In collaboration with domain experts, different architectural concepts and allocation

strategies may be analyzed, taking into consideration performance and safety requirements that were captured during the requirements analysis phase.

In the subsequent subsystem architectural design phase, decisions are made on which operational contracts within a physical subsystem should be implemented in hardware and which should be implemented in software (hardware/software trade-off analysis). The different design concepts are captured in the deployment model and verified through regression testing.

The deployment model defines the system architecture baseline for the subsequent hardware/software (HW/SW) development. Essential documents that are generated from the deployment model are the following:

- HW/SW design specifications.
- Logical interface control document (ICD).

The outlined systems engineering process is model-based, using the UML 2.0 as a modeling language. The essential UML artifacts are the following:

- Use-case diagram.
- Activity diagram.
- State-chart diagram.
- Sequence diagram.
- Structure diagram.

The following sections detail the workflow in the different systems engineering phases with examples from a case study.

Requirements Analysis

Requirements Capture

The requirements analysis phase starts with the analysis of the process inputs. Customer requirements are translated

into a set of requirements that define what the system must do (functional requirements) and how well it must perform (quality of service requirements). The captured requirements are imported into the model/requirements repository.

Definition of Use Cases

Once the requirements are sufficiently understood, they are clustered in use cases. A use case describes a specific operational aspect of the system (operational thread). It specifies the behavior as perceived by the users and the message flow between the users and the use case. It does not reveal or imply the system's internal structure (black-box view).

A set of use cases is grouped in a use-case diagram. Use cases can be structured hierarchically. There is no *golden rule* with regard to the number of use cases needed to describe a system. Experience shows that for large systems, typically six to 24 use cases are defined at the top level. At the lowest level, a use case should be described by at least five with a maximum of 25 essential use-case scenarios.

At this stage, emphasis is put on the identification of *sunny day* use cases, assuming an error/fail-free system behavior. Exception scenarios are identified at a later stage (\Rightarrow system functional analysis) through model execution. If more than five error/fail scenarios are found for a use case, a separate exception use case will be defined and added to the sunny day use case via the *include* or *extend* relationship.

Figure 3 shows the use-case diagram of the case study. The system is a main battle tank. Two use cases are studied:

- Acquire target.
- Engage target.

The actors are commander and gunner.

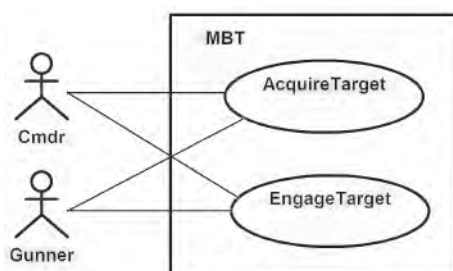
The use-case diagram is imported into the model/requirements repository. The use cases are then linked to the system requirements and checked for complete coverage.

System Functional Analysis

In the system functional analysis phase, each use case is translated into a model and the underlying requirements then verified and validated through model execution. A message-driven approach (Figure 4) is used. Characteristics of this approach are the following:

- The system structure is described by means of a UML 2.0 structure diagram using blocks as basic structure elements and ports, and the notation of *provided/required* block interfaces.
- Communication between blocks is based on messages (*service requests*).

Figure 3: Case Study Use-Case Diagram



- System functionality is captured through operational contracts (*services*), e.g., operation1(),..., operation4() in Figure 4.
- Functional decomposition is performed through decomposition of operational contracts.

The black-box use case analysis starts with the definition of the use case model context diagram. The UML 2.0 artifact used for this is the structure diagram. Elements of this diagram are blocks representing the actors and the system.

Identifying the black-box use-case scenario is the next analysis step. A use-case scenario describes a specific path (functional flow) through a use case. It details the message flow between the actors and the use case and the resulting behavior (operational contracts) of the recipient. In the UML, a scenario is graphically represented in a sequence diagram. The lifelines in the black-box sequence diagram are the actors and the system (see Figure 5).

Once a set of essential scenarios is captured, the identified functional flow information is merged into a common use case description. The UML artifact used for this is the activity diagram (see Figure 6). Each action block in this diagram corresponds to an operational contract in a sequence diagram. The black-box activity diagram plays an essential role in the upcoming architectural design phase.

Based on the information captured in the black-box sequence diagrams and black-box activity diagram, the blocks of the black-box use case model next are populated, and ports and associated interfaces are defined. Figure 7 (see page 20) shows the resulting static model of the use case Acquire Target.

The next step in black-box use case analysis is the description of system-level, state-based behavior. The UML artifact used for this is the state-chart diagram (see Figure 8 on page 20). State-charts are hierarchical state machines that visualize system states and modes and their changes as the response to external stimuli. The use case related state-based behavior is derived from the captured use-case scenarios.

At this stage, the verification and validation (V&V) of the black-box use case model and the underlying requirements (i.e., operational contracts) can start. V&V is performed through model execution using the captured black-box use-case scenarios as the basis for respective stimuli. It should be noted that following the previously outlined key objectives of this process, the focus is on the analysis of the generated sequences rather than on the underlying functionality.

So far, the use-case model represents an error/fail free (*sunny-day*) behavior. At this stage, it may be extended regarding possible exceptions (*rainy-day* behavior).

The outlined steps are repeated for each use case, except that instead of creating new activity diagrams, the initial black-box activity diagram incrementally is extended based on the information of the new use-case scenarios. The same applies to the system black-box state-chart diagram. The subsequent model V&V is performed in two steps: First, the extended black-box system model is verified/validated through model execution using the black-box use-case scenarios as the basis for respective stimuli. Then, the collaboration of the implemented use cases is verified through regression testing.

At the end of the functional analysis phase, a black-box system model is built of verified and validated operational contracts, representing the underlying functional requirements. The black-box system model is imported into the model/requirements repository and the operational contracts are linked to the high-level system requirements. The black-box use-case scenarios are imported into the test data repository for reuse in the subsequent architectural design phases.

Architectural Design

System Architectural Design

Focus of the system architectural design phase is the allocation of the verified and validated operational contracts to a physical architecture. The allocation is an iterative process and is performed in collaboration with domain experts. Different architectural concepts and allocation strategies may be analyzed, taking into consideration performance and safety requirements that were cap-

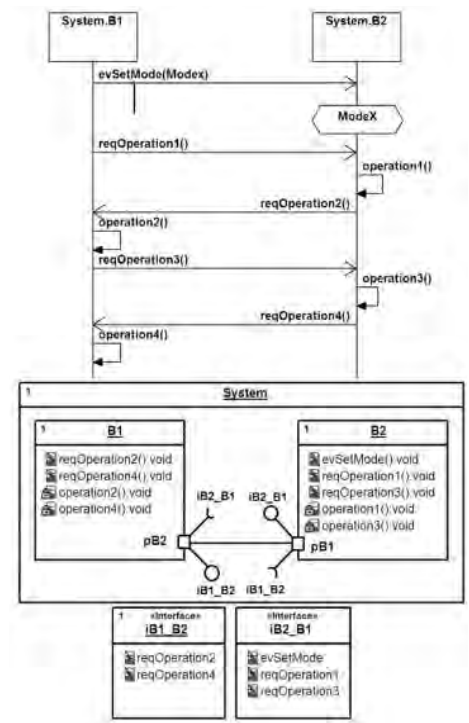


Figure 4: Message-Driven Modeling Approach

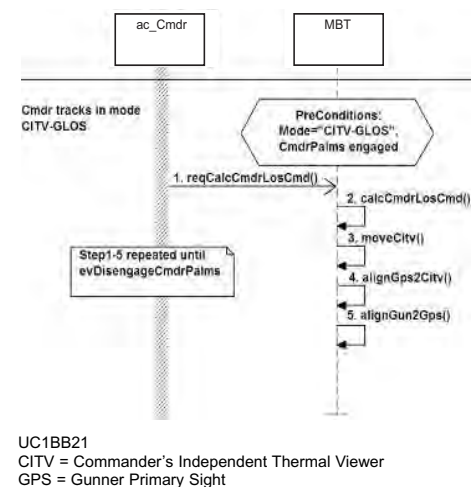


Figure 5: Black-Box Use-Case Scenario

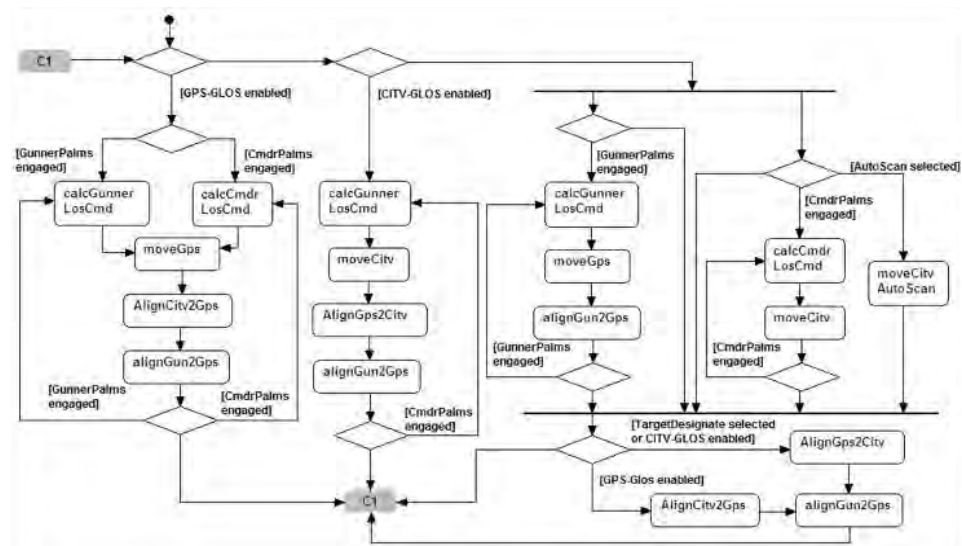


Figure 6: Black-Box Activity Diagram (Use Case Acquire Target)

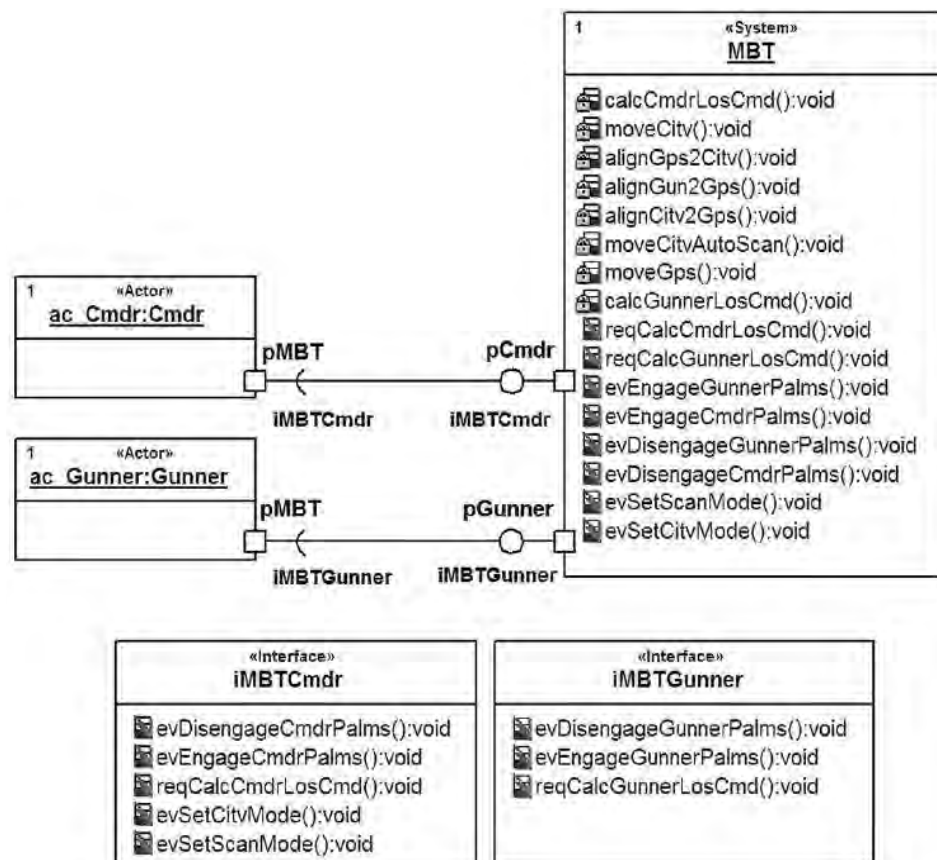


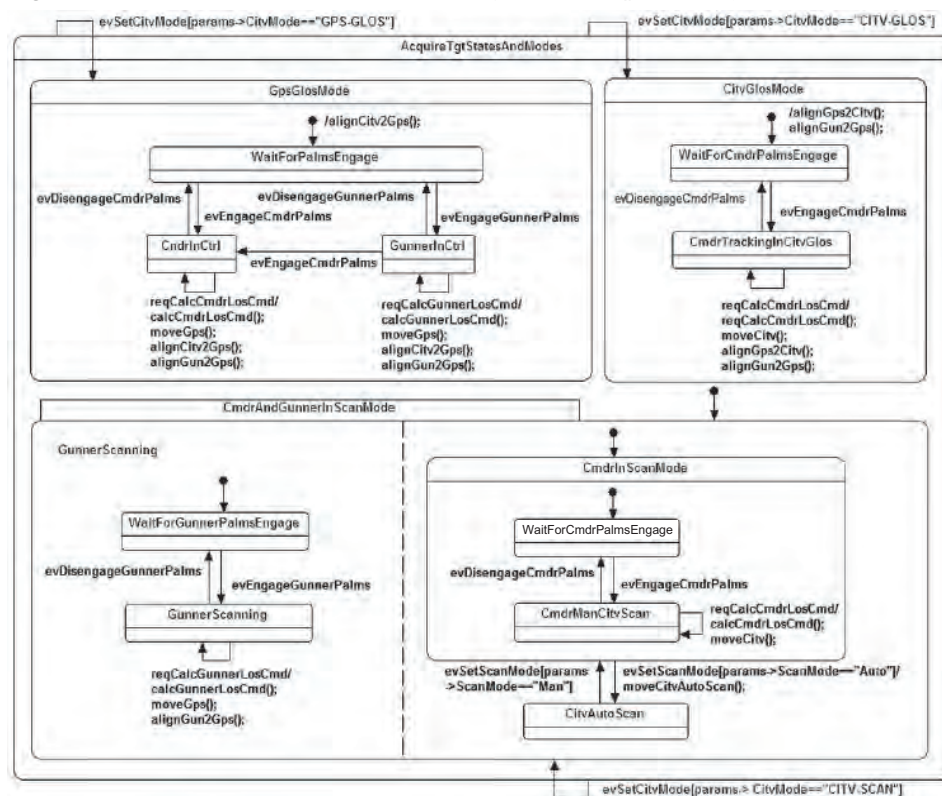
Figure 7: Static Black-Box System Model (Use Case Acquire Target)

tured during the requirements analysis phase.

System architectural design starts with the definition of the physical subsystems. The UML 2.0 artifact used for this is the

structure diagram. Constituents of this model are the actor blocks and the system block. Parts of the system block are the physical subsystems of the chosen architecture. In the case study, the system

Figure 8: Black-Box System State-Based Behavior (Use Case Acquire Target)



design consists of six physical subsystems (LRU = line replaceable unit, see Figure 9A).

Next, the previously identified black-box operational contracts are allocated to the physical subsystems using an activity diagram (white-box activity diagram). Essentially, this activity diagram is a copy of the black-box activity diagram. The only difference is that the system now is partitioned into swim lanes, each representing a physical subsystem. Based on the chosen design concept, the system operational contracts are *moved* to respective subsystem swim lanes. An essential precondition for this allocation is that the initial links (functional flow) between the operational contracts are maintained.

The white-box activity diagram is complemented by the definition of white-box sequence diagrams. White-box sequence diagrams are decompositions of the previously captured black-box sequence diagrams and are utilized to identify the interfaces of the physical subsystems. In white-box sequence diagrams, the system lifeline is split into a set of subsystem lifelines. Based on the allocation defined in the white-box activity diagram, the subsystem operational contracts are placed on respective subsystem lifelines. To maintain the initial functional flow, service requests from one physical subsystem to the other may need to be generated. They define the interfaces between the subsystems.

In the example shown in Figure 10, the implementation concept was that LRU1 was considered the commander's input/output device. Most of the identified functionality had to be implemented in LRU5. The Commander's Independent Thermal Viewer control had to be in LRU6. In this scenario, LRU4 served as a gateway between LRU1 and LRU5. By mapping the use case scenario to the physical architecture, the links and the associated ports and interfaces are defined for each involved physical subsystem. For each physical subsystem, the associated state-based behavior is captured in a state-chart diagram. These state-chart diagrams will extend incrementally with each mapped use-case scenario.

The outlined process is performed iteratively for all black-box scenarios. Figure 9A shows the final result. Figure 9B depicts for the chosen architectural design the resulting physical subsystem interfaces by means of an N-squared (N^2) chart. An N^2 chart is structured by locating the nodes of communication on the diagonal, resulting in an $N \times N$ matrix for a set of N nodes. For a given node, all out-

puts (UML 2.0 *required interfaces*) are located in a row of that node and inputs (UML 2.0 *provided interfaces*) are in the column of that node.

The correctness and completeness of the system architecture model is checked through model execution. Once the model functionality is verified, the architectural design can be analyzed with regard to the performance and safety requirements. The analysis typically includes failure modes effects analysis and mission criticality analysis.

Subsystem Architectural Design

This phase focuses on the implementation of the allocated operational contracts. Decisions are made on which operational contracts in a physical subsystem should be implemented in hardware (mechanical/application-specific integrated circuit) and which should be implemented in software. For operational contracts that span more than one domain, further analysis will be needed. Subsystem domain experts may participate in this analysis.

Once the HW/SW design decisions are made, the workflow is similar to the one outlined in the system architectural design phase. In each white-box use-case scenario, the physical subsystem lifelines are split into HW and/or SW lifelines, each representing a subsystem component. Based on the chosen HW/SW design concept, operational contracts then are placed on respective subsystem component lifelines, and the associated functional flow between subsystems and subsystem components established through respective service requests (see Figure 11 on page 22). Thus, subsystem component ports and interfaces are defined. For each physical subsystem component, the associated state-based behavior is captured in a state-chart diagram. These state-chart diagrams will extend incrementally with each white-box use-case scenario.

The outlined process is performed iteratively for all white-box scenarios. The final deployment architecture is verified through regression testing.

At the end of the system architectural design phase, the deployment model is imported into the model/requirements repository, and the HW/SW assigned operational contracts are linked to the original requirements. For each physical subsystem the following documents are generated from the deployment model as handoffs to the subsequent hardware and software design:

- HW/SW design specification.

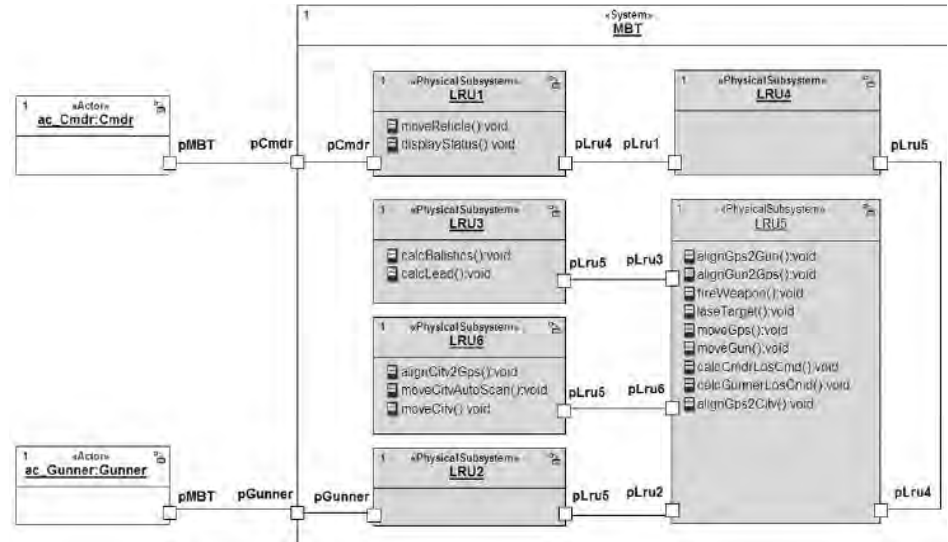


Figure 9A: Activity Operational Contracts Allocated to an LRU Network Architecture

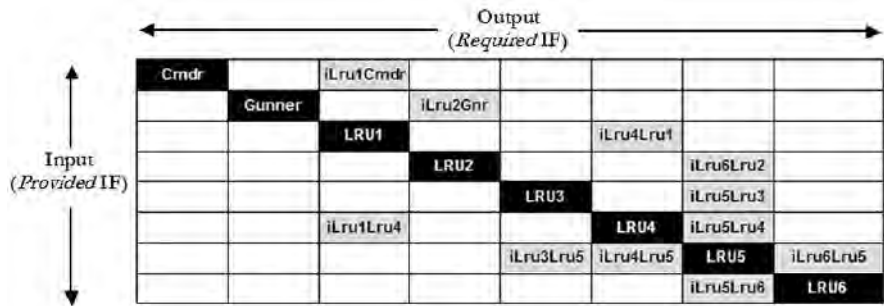


Figure 9B: Documentation of Physical Subsystem Interfaces by Means of an N^2 Chart

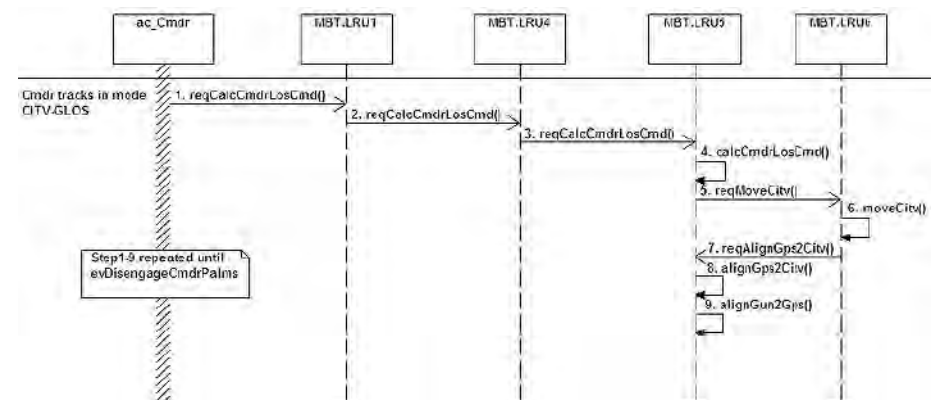
- Logical interface control document (N^2 chart).
- Subsystem/subsystem component test vectors derived from the system-level use-case scenarios.

Conclusion

For a long time, the UML was considered a modeling language suitable only for software developers that follow the object-oriented paradigm. This article demonstrates that with the release of UML 2.0 – specifically with the introduction of (composite) structure diagrams – the UML could also be applied to function-driven systems engineering.

Based on this common, paradigm-independent language for both systems engineers and software engineers, an integrated systems/software development process could be defined, allowing a seamless transition between the two domains. Still, the UML needs some extensions to cover the needs of systems engineers completely (e.g., time-continuous communication). For this purpose, the object management group formed the Systems Modeling Language (SysML) consortium [2]. The first release of the SysML specification will be in the fourth quarter 2005. ♦

Figure 10: White-Box Scenario UC1WB21 (Decomposed Black-Box Scenario of Figure 5)



CROSSTALK

The Journal of Defense Software Engineering

Get Your Free Subscription

Fill out and send us this form.

309 SMXG/MXDB

6022 FIR AVE

BLDG 1238

HILL AFB, UT 84056-5820

FAX: (801) 777-8069 DSN: 777-8069

PHONE: (801) 775-5555 DSN: 775-5555

Or request online at www.stsc.hill.af.mil

NAME: _____

RANK/GRADE: _____

POSITION/TITLE: _____

ORGANIZATION: _____

ADDRESS: _____

BASE/CITY: _____

STATE: _____ ZIP: _____

PHONE: (____) _____

FAX: (____) _____

E-MAIL: _____

CHECK BOX(ES) TO REQUEST BACK ISSUES:

JULY2004 ☐ TOP 5 PROJECTS

AUG2004 ☐ SYSTEMS APPROACH

SEPT2004 ☐ SOFTWARE EDGE

OCT2004 ☐ PROJECT MANAGEMENT

NOV2004 ☐ SOFTWARE TOOLBOX

DEC2004 ☐ REUSE

JAN2005 ☐ OPEN SOURCE SW

FEB2005 ☐ RISK MANAGEMENT

MAR2005 ☐ TEAM SOFTWARE PROCESS

APR2005 ☐ COST ESTIMATION

MAY2005 ☐ CAPABILITIES

JUNE2005 ☐ REALITY COMPUTING

JULY2005 ☐ CONFIG. MGT. AND TEST

AUG2005 ☐ SYS: FIELDG. CAPABILITIES

SEPT2005 ☐ TOP 5 PROJECTS

OCT2005 ☐ SOFTWARE SECURITY

TO REQUEST BACK ISSUES ON TOPICS NOT LISTED ABOVE, PLEASE CONTACT <STSC.CUSTOMERSERVICE@HILL.AF.MIL>.

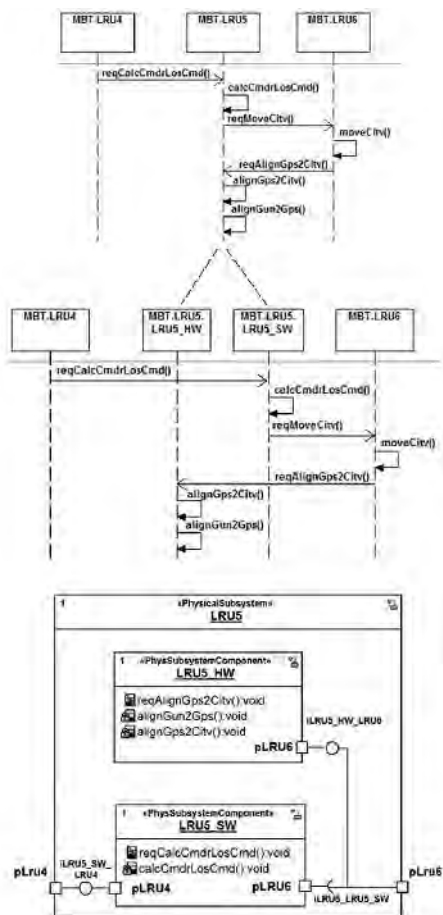


Figure 11: Subsystem Architectural Design

References

1. Object Management Group. Unified Modeling Language. "UML 2.0 Specification" 17 Aug. 2005 <www.uml.org/#UML2.0>.
2. SysML Forum. "SysML Specification Vers. 0.9 Draft" 10 Jan. 2005. 17 Aug. 2005 <www.sysml.org/artifacts.htm>.

About the Author



Hans-Peter Hoffmann, Ph.D., is director and chief methodologist for Systems Design at I-Logix Inc., a real-time object-oriented and structured systems design automation tool vendor. Focusing on methodology consulting, Hoffman works as an international consultant for model-based system development. He has 25 years experience in the design and development of complex systems in the aerospace/defense and automotive industries. Hoffmann co-developed the I-Logix Integrated Systems/Software Development Process HARMONY, which combines Unified Modeling Language (UML)/Systems Modeling Language-based systems engineering and UML-based software engineering. Previously as director of the simulation department of the Missile Division at Messerschmitt-Bölkow-Blohm Germany, (now European Aeronautic Defense and Space Company N.V.), he developed a methodology for modeling and analysis of flight control systems.

I-Logix Inc.

3 Riverside DR

Andover, MA 01810

Phone: (978) 645-3022

Fax: (978) 682-5995

E-mail: peterh@ilogix.com

WEB SITES

International Association of Software Architects

www.iasarchitects.org

The International Association of Software Architects (IASA) is a nonprofit organization dedicated to the advancement and sharing of issues related to software architecture in the enterprise, product, education, and government sectors. IASA functions as an umbrella organization to chapters spread throughout the world, and as a driving force for research and standards that advance the understanding of software architecture. IASA helps sustain city chapters by forming software architecture user groups and by fostering the sharing of ideas and information between these city groups and industry leaders.

mation between these city groups and industry leaders.

International Enterprise Architecture Center

www.ieac.org

The International Enterprise Architecture Center (IEAC) is a global, independent, nonprofit professional membership organization dedicated to the dissemination of knowledge about enterprise architectures, management, and the marketplace to help members strengthen their performance and better serve society. The IEAC assists in the development of enterprise architecture frameworks, methods and disciplines across all enterprise environments.